IF:
a Tool-set for validation of distributed real-time systems

Susanne Graf

Marius Bozga, Laurent Mounier,
Yassine Lakhnech, Joseph Sifakis

Grenoble
Theory, methods and tools for design and validation of distributed and safety critical systems

• Synchronous languages, development of embedded systems
  – Lustre language: compilation, verification and test
    → Telelogic SCADE

• Tools and methods based on timed and hybrid automata
  – synthesis and validation of schedulers and controllers
  – Kronos tool for the verification of timed systems

• Tools and methods for communication systems
  – Semantics and real-time extensions of design languages
  – Verification of security protocols
  – Validation tools: Xesar, CADP, TGV, Invest, IF
**Motivation**

**Goal**

Combine state-of-the-art validation with commercial development tools

**Context**

Telecommunication systems, Real-time embedded systems
Model-checking: its problems

The idea: why MC is attractive

- design description
- fully automatic
- check fully automatic
- semantic Model

The reality: why has MC a bad reputation

- abstract design description
- fully automatic
- Model
- all properties hold!
- what does this prove?
- state explosion
- too bad!
- exploitable model
- valuable results

or

- detailed design description
- fully automatic
- a lot of hard handwork
- exploitable model
- valuable results
Model-checking: how it should be

This bad reputation must not be justified

- detailed design description
- A designer friendly tool
- exploitable model

valuable verification results, abstract models, test cases or counter examples
Principle of a validation environment

Extract parts, Optimization, Abstraction, Compositional Methods, Environment constraints

Design Language

Development Environment

UML, SDL,..

IF

structured representation

transl

Model

LTS state explosion

Model-checking and Diagnostics tools

feedback
Need for a structured system representation

1. Intermediate and tool exchange format
   - Basis for static analysis, abstraction and compositional methods
   - Connection of a large range of high level design languages of with analysis tools (model-checking, performance,...)
   - Exchange of structured system descriptions between analysis tools

2. Study of time models
   - Need for a appropriate time extensions of languages for communicating and distributed systems (SDL, UML)
   - Appropriate for design and verification of real-time systems
Outline

1. Motivations
2. IF intermediate representation
3. IF validation tool-set
4. Case studies
5. Conclusions
2. IF intermediate representation

System structure at instant $t$

Communicating extended timed automata (with urgency)

Communication/Interaction
- asynchronous channels
  (reliable?, bounded?, delay?)
- synchronous rendez-vous
- shared variables

Time representation
Timed automata with
Urgency of transitions
(eager, lazy, delayable)
IF: Processes

- A set of **local variables**
  - elementary: bool, int, ... timers and clocks, ...
  - structured: array, record
  - abstract

- A set of **control states** with attributes:
  - stable/nostable (control observable states)
  - save and discard sets (reordering of input message buffer)

- A set of **control transitions**:
  \[
  s \xrightarrow{\text{guard} \rightarrow \text{input} ; \text{body}} \Rightarrow s' \]
  urgency ; priority
IF: Transitions (abstract syntax)

- **guard**: boolean expression on data, timers, clocks
- **input**: asynchronous message inputs from buffers
- **sync**: gate synchronization
- **body**: action*
  - asynchronous message outputs to buffers
  - re/setting of timers/clocks
  - assignments
  - complex instructions
- **urgency attribute**: eager, lazy, delayable
- **priority**
Timed automata with urgency
[BornotSifakis97]

- System transitions take 0 time
  (assimilated with an event "transition started", "transition terminated", ...) & time progresses in states, measured by clocks and timers

- **Urgency** defines when enabled system transitions are taken
  - enabled **eager** transitions are urgent, that is terminated « now » (or disabled by other system transitions)
  - enabled **lazy** transitions are never urgent, that means they can be disabled by time-progress
  - enabled **delayable** transitions are not disabled by time-progress, but it is taken for granted that they will be taken (except if disabled by other system transitions)
Timed automata with urgency

Allow to express a rich spectrum of time paradigms

1. Totally asynchronous view (no assumption on time progress): all transitions are lazy
   Ensure safe behaviour despite violation of deadlines

2. Synchronous view (next tick/input when system has finished): all transitions are eager
   Ensure safe behaviour under strong assumptions (risk of time-lock)

3. Real-time views: different urgency types and time guards:
Outline

1. Motivations
2. IF intermediate representation
3. IF validation tool-set
4. Applications
5. Conclusion and perspectives
3. Architecture of the IF tool-set

- ObjectGEODE
- SDL
- IF
- sdl2if
- if.open
- Static Analysis
- environment comp
- invariant generation
- abstraction (InVeSt)
- ...
Translation from SDL to IF: sdl2if

• Based on an ObjectGeode API
  
  we follow standard evolution of SDL

• Supports almost all of SDL'96:
  
  - timeouts are translated by time-guards
  - elimination of block hierarchy ("flat" architecture)
  - destination of outputs is statically determined if possible
    (only delaying channels represented explicitly)
  
  Only for more efficient verification
  
  - procedures are inlined (no recursion allowed)
Translation IF to LTS

Simulator construction: if.open

- implements:
  - discrete/dense time
  - partial order reduction
  - compositional generation

- supports:
  - random/guided simulation
  - on-the-fly verification
  - explicit LTS construction
LTS level validation components

• Basic Functionalities
  - switch representations
  - parallel composition
  - draw graphical representations (valid property)
  - generate MSCs from (diagnostic) sequence (invalid property)

• Model-checking:
  • temporal-logic properties (Evaluator, Kronos)
  • behavioral comparison and reduction (Aldebaran)
    (both including diagnostic capabilities)

• Test case generation (TGV)
**Static Analysis and Abstraction**

**PRINCIPLE**

- Source code transformation in order to get
  - a smaller state representation
  - less states, which represent sets
- **Preserve a set of (safety) properties** (strongly or weakly)

- Combine several static analysis methods
Static Analysis and Abstraction (property independant)

- Reset all live variables not live in some control point (its value is irrelevant in this state)
- Invalidate non-live clocks (clock reduction)
- Eliminate globally dead variables
- Replace constants by their value

Live variable analysis and constant elimination
Static Analysis and Abstraction (property dependant)

- Eliminate non relevant parts of the system with respect to a slicing criterion (variables, messages, transitions, processes)

observables: messages, variables, … (in particular control states)
test purpose, abstract behaviour, temporal logic formula…

example
Static Analysis and Abstraction (property dependant)

IF \rightarrow abs \rightarrow IF

property under check

abstraction rel.

abstraction

(variable elimination, data abstraction, predicate abstraction (InVeST), …)

(test purpose, observer, TL property…)

----- Stenungsund, July 5, 2001 -----
Static Analysis and Abstraction

Summary

• In practice: drastic reductions of the state graph
• "abstract program" computed, can be directly used by other tools
• Notice:
  - static analyses and abstractions can be combined, preserving the intersection of the properties
  - abstraction means (in general) weak preservation of properties
Architecture of the IF tool-set

ObjectGEODE
specification design

SDL

Static Analysis
live
clock reduction
slice
invariant generation
abstraction (InVeSt)
...

feed back s.a.

sdl2if

IF

if.open

EUT

if2pml

Liège

if2lash

LASH

Promela
Spin

Evaluator
µ-calculus
checker

|| comp

draw

Aldebaran
compare
minimize

TGV
Test case
generation

LTS

explicit
(aut)

symbolic
(SMI)

implicit
C succ
function

...
1. Motivations

2. IF intermediate representation

3. IF validation and test generation environment

4. **Applications**

5. Conclusion and perspectives
Validation methodology

- validation methodology
- live analysis, dead code elimination, variable elimination
- on-the-fly verification, guided simulation, deadlock detection...
- slicing, abstraction...
- model-checking, test generation, ...
- diagnostic: MSC, abstract LTS
- model generation + p.o.
- advanced static analysis
- model exploration + p.o.
- basic static analysis
- IF Spec
- SDL Spec
- Environment
- Requirements

----- Stenungsund, July 5, 2001 -----
Validation methodology: taking into account environment constraints

Open systems: environment constraints (EC) are essential for successful verification of verification results.

\[
\text{Sys} \models \text{EC} \Rightarrow P
\]

• Solution: describe EC by a (set of) processes \( E \)

Verify the \( \text{Sys} \parallel E \),

where \( \text{Sys} \) and \( E \) communicate by synchronous rendezvous.
Environment constraints:
- $E$ sends requests $s$ only if $x=0$
- $E$ responds $res(y)$ iff $Sys$ has sent $req(x)$ and $y < x+5$
Applications

- **ATM adaptation layer transport protocol (SSCOP)**
  - live analysis, weak bisimulation minimization
  - state size: 2000B → 100B
  - unexplorable → 1,000,000 states

- **Medium access for wireless ATM (Mascara)**
  - live analysis, slicing, μ-calculus checking

- **Ariane-5 flight controller** (40 minutes of flight)
  - description obtained by reengineering
  - many timers (smallest with 70ms rate)
  - 31 SDL processes
Mascara Protocol

- Verification case study of Esprit-LTR Vires project
- Medium Access Control protocol for wireless ATM
  ⇒ mediation between access points and mobile terminals

ATM layer

Mascara Adaptation Layer for Wireless Comm

Control
Error Control
Data Transmission

Radio Transmission layer
Mascara Dynamic Control

- Set up and release associations and virtual connections (address mapping, resource management)

- 8 SDL processes + environment

Medium size protocol: 10 000 lines of textual SDL

⇒ complex data structures, large number of messages and potentially interacting protocols
Mascara: modeling choices

Environment and Requirements

1. unrestricted environment → queues of unbounded length
   - restrict the number of requests per time unit

2. a priori no functional environment restrictions and no requirements given
   - start with simple properties and chaotic environment and strengthen as much as possible/necessary

Expression of requirements

- temporal logic
- abstract behaviors in terms of LTS: comparison modulo (bi)simulation or computation of exact property modulo some observation criterion
Expression of Requirements

Example: « each association-request will be confirmed »

most general

non regular

regular approximation

we compute

the exact property satisfied by the system

- a_req
- a_conf*
- a_req
- a_conf*
- a_req
- a_conf*
Mascara: verification strategies

Direct generation failed even using all optimizations

Use of a compositional approach:

- **Compositional generation**
  - generate and minimize the LTS associated to each process
  - apply parallel composition at the LTS level
- **Compositional verification**
  - split a global property into a set of local properties
  - verify each local property using an abstract environment

In combination with:
- static analysis
- partial order reduction
**Mascara: Complexity results**

<table>
<thead>
<tr>
<th>ent</th>
<th>n°</th>
<th>method</th>
<th>states</th>
<th>reduc</th>
<th>trans</th>
<th>redu</th>
<th>time</th>
<th>redu</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP</td>
<td>1</td>
<td>no reduction</td>
<td>7 000 K</td>
<td>-</td>
<td>30 000 K</td>
<td>-</td>
<td>3h</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>p.o.</td>
<td>900 K</td>
<td>8</td>
<td>1 800 K</td>
<td>17</td>
<td>37m</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>live reduction</td>
<td>400 K</td>
<td>17</td>
<td>1 500 K</td>
<td>20</td>
<td>12m</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>p.o. + live</td>
<td>28 K</td>
<td>250</td>
<td>52 K</td>
<td>577</td>
<td>1m52</td>
<td>118</td>
</tr>
<tr>
<td>MT</td>
<td>5</td>
<td>no reduction</td>
<td>4 300 K</td>
<td>-</td>
<td>12 000 K</td>
<td>-</td>
<td>2h51</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>p.o.</td>
<td>3 100 K</td>
<td>1.3</td>
<td>7 400 K</td>
<td>1.6</td>
<td>1h30</td>
<td>1.9</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>live reduction</td>
<td>63 K</td>
<td>68</td>
<td>325 K</td>
<td>36</td>
<td>1m03</td>
<td>162</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>p.o. + live</td>
<td>6 K</td>
<td>716</td>
<td>20 K</td>
<td>600</td>
<td>7s</td>
<td>1460</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>live + po + slice</td>
<td>1 K</td>
<td>4300</td>
<td>3 K</td>
<td>4000</td>
<td>4s</td>
<td>2550</td>
</tr>
<tr>
<td>all</td>
<td>10</td>
<td>live + p.o.</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>all</td>
<td>11</td>
<td>4_{min}</td>
<td></td>
<td>8_{min}</td>
<td>218 K</td>
<td>1 140 K</td>
<td>n.a.</td>
<td></td>
</tr>
</tbody>
</table>
Conclusions: method for design validation

Commercial Design tool

High Level Design Reference Model
non-determinism: +++
details: -/+  
refinement / compilation

Target
non-determinism: 0
details: +++

IF tool box

Environment & Properties
constraints and assumptions

constraint system
non-determinism: +
details: -/+  
compile together
test case generation

test cases

analysable model

all tricks: s.a. comp. p.o.

feedback

Similar approach for performance evaluation
Tool Perspectives

- **dynamic** features are needed:
  - for connection with UML, JAVA, ...
  - for connection with symbolic validation tools
    definition of *dynamicIF*

- more general annotations of type `assume/assert` for requirement expression and test case generation

- more static analysis, abstraction and constraint propagation:
  connection with PVS based *InVest* tool

- more compositional verification methods

- better diagnostic facilities

- **Connections:**
  - connection with ASM tools
  - connection with performance evaluation tools
http://www-verimag.imag.fr/~async
Time and system progress in simulation

- decomposition
- abstraction
- combined time and system steps

execution: independent time dynamics

start a
end a

correct approximation:
- decomposition
- abstraction

action a

system progress
Time and system progress in simulation

During simulation/validation:

• Problem: how to decide the time point of the next event: now? or should time progress, and how much?

Time progress must depend on assumptions made by the designer
Slicing: example

```
var: u,w,x,y,z
```

Slicing criterion:

- Observable events: in2, out3
  
  \[ \text{var: } u,x,z \]

- Environment: in2, in3, in4
  
  \[ \text{var: } x,z \]
Slicing: example

Slicing criteria:

- observable events: `in2`, `out3`
  
  ```
  var: u, x, z
  ```

- environment: `in2`, `in3`, `in4`
  
  ```
  var: x, z
  ```

- weak bisimulation reduction

```plaintext
env!out3(x+z)
```

```plaintext
q?in2(z)
```

```plaintext
x := 2+z
```

```plaintext
q?in3(z)
```

```plaintext
var: x, z
```
clock $x$;

$1 \leq x \leq 3$
Timed automata with urgency 2

$q_0 \rightarrow q_1$

$1 \leq x \leq 3$

urgency

eager

$\begin{array}{c}
q_0 \\
q_1
\end{array}$

1 2 3

x

delayable

$\begin{array}{c}
q_0 \\
q_1
\end{array}$

1 2 3

x

lazy
Timed automata with urgency 2

$1 \leq x \leq 3$

urgency
Timed automata with urgency 2

- **Eager**
  - Invariant: \( x < 1 \) \( v \) \( x > 3 \)
  - States: \( q_0, q_1 \)
  - Transitions:
    - \( q_0 \) to \( q_0 \) (\( x = 1 \))
    - \( q_0 \) to \( q_1 \) (\( x = 2 \))
    - \( q_1 \) to \( q_1 \) (\( x = 3 \))

- **Lazy**
  - Invariant: true
  - States: \( q_0, q_1 \)
  - Transitions:
    - \( q_0 \) to \( q_0 \) (\( x = 1 \))
    - \( q_0 \) to \( q_1 \) (\( x = 2 \))
    - \( q_1 \) to \( q_1 \) (\( x = 3 \))

- **Delayable**
  - Invariant: \( x < 3 \) \( \& \) \( x > 3 \)
  - States: \( q_0, q_1 \)
  - Transitions:
    - \( q_0 \) to \( q_0 \) (\( x = 1 \))
    - \( q_0 \) to \( q_1 \) (\( x = 2 \))
    - \( q_1 \) to \( q_1 \) (\( x = 3 \))

- States:
  - \( q_0 \)
  - \( q_1 \)

- Transition conditions:
  - \( 1 \leq x \leq 3 \)

- Urgency

---

Stenungsund, July 5, 2001